A Computational Array for the QR-Method

Lennart Johnsson
California Institute of Technology
Pasadena, California 91125

5019:TM:82

(Presented at the Conference on Advanced Research in VLSI,
January 25 - 27, 1982, Massachusetts Institute of Technology)

A COMPUTATIONAL ARRAY FOR THE QR-METHOD

LENNART JOHNSSON

Computer Science
California Institute of Technology
Pasadena, CA. 91125

ABSTRACT

The QR-method is a method for the solution of linear system of equations. The matrix R is upper triangular and Q is a unitary matrix. In equation solving Q is not always computed explicitly. The matrix R can be obtained by applying a sequence of unitary transformations to the matrix defining the system of equations. Householder's method or Given's method can be used to determine unitary transformation matrices. This paper describes a concurrent algorithm and corresponding array for computing the triangular matrix R by Householder transformations. Particular attention is given to issues such as broadcasting and pipelining.

INTRODUCTION

In this paper algorithms for the solution of systems of linear equations by the QR-method are described. In particular algorithms for the computation of the factorization through Householder transformations are investigated. Special attention is given to implementation issues in VLSI technology.

As the feature sizes are scaled down the electrical characteristics of the circuitry changes. These changes have many implications, also algorithmically.

The research presented in this paper is supported by the Defense Advanced Research Project Agency under contract N00014-79-C-0597 with the California Institute of Technology

The switching speed of devices increases but wire delay does not decrease. For a wire the length of which scales down with the scaling factor, the delay stays constant under scaling, while for a wire of constant length the delay actually increases with the square of the scaling factor. Hence, it is of particular importance to study the communication needs of algorithms intended for implementation in a technology with feature sizes of one micron or less. Algorithms with a high degree of concurrency and which only requires communication with computational elements that can be arranged to be nearest neighbors in a plane are preferable.

Broadcasting typically implies a performance penalty in VLSI technology. Pipelining can be used to eliminate broadcasting and in general improve the computational rate. If the data flow through the array is turbulent, i.e., the data flows through loops in the network, it is in general necessary to space the data out in time due to the pipelining, as demonstrated in [Johnsson 81] for arrays for Gaussian elimination. Furthermore, pipelining and partial pivoting do not combine nicely. Partial pivoting makes the computational order data dependent. It will typically be the case that a computational element has to be idle for some time until a new order of computation can be determined and the computations resumed at the proper location in the array. A linear pipelined array is as efficient as a two-dimensional array for Gaussian elimination with partial pivoting, [Johnsson 81].

H. T. Kung [Kung 80a] has presented algorithms for, for instance, matrix multiplication and LU-decomposition, using systolic, hexagonal, arrays. S. Y. Kung [Kung 80b] has studied LU-decomposition on two-dimensional arrays of processors arranged as a regular mesh with interconnections to six nearest neighbors (hexagonal interconnect, optimal for Gaussian elimination). The LU-decomposition is performed by Gaussian elimination. Both in [Kung 80a] and [Kung 80b] the arrays are assumed to be large enough to hold the "entire" problem, i.e., the entire matrix for a full matrix problem and a window corresponding to the bandwidth for a band matrix. The computations are fully instantiated in space and the control of the data flow implicit in the interconnection of the processors. Even though several processors may be placed on a chip as the feature sizes are scaled down there will certainly be a need to solve problems of a size larger than what corresponds to the number

of processors in a system. Many problems have to be partially instantiated in time, partially in space. Johnsson [Johnsson 81] has modified the naive arrays and introduced explicit control to allow for partial instantiation in time and also studied the effect of pipelining on the data rate.

The QR-method for the solution of linear system of equations (and eigenvalue problems) is based on a sequence of unitary transformations. Using unitary transformations guarantees numeric stability, [Wilkinson 65]. The two most common methods used to determine unitary transformation matrices are Given's method of plane rotations and Householder's method based on Hermitian unitary matrices. The error analysis carried out by Wilkinson [Wilkinson 65] gives lower upper bounds for Householder's method than for Given's method. Given's method does not necessarily require evaluation of square roots as shown by, e.g., Gentleman [Gentlemen 73] and Hammarling [Hammarling 74]. Both Householder's method and Given's method have data independent data flow. Systolic arrays for Given's method have been described by Heller and Ipsen in the report [Heller 81] and in a paper at this conference.

In this paper concurrent algorithms for the QR-method based on Householder transformations are studied. Particular emphasis is placed on implementation issues such as broadcasting, pipelining, explicit control and partitioning of a problem for an array that does not hold the "entire" problem.

HOUSEHOLDER TRANSFORMATIONS

Descriptions of the QR-method and Householder transformations can be found in, e.g., [Wilkinson 65], [Isaacson 66], and [Dahlquist 74].

If the system of equations to be solved is written

$$Ax = y$$

then the QR-method consists in factorizing the matrix A into an upper triangular matrix R and a unitary matrix Q such that $A = QR$. The system of equations can be solved as follows

$$QRx = y$$

$$Rx = Q^{-1}y = z$$

$$x = R^{-1}z$$

Since R is upper triangular the last step is solved by back substitution and not by explicitly computing the inverse. Using Householder transformations to compute the matrix R give Q^{-1} as the product of the transformation matrices.

$$P_{n-1}\cdots P_1 Ax = P_{n-1}\cdots P_1 y$$

or

$$Rx = Q^{-1}y$$

where P is of the form $(I-2ww^*)$ and I is the identity matrix.

For each application of a matrix P the elements below the diagonal in one column are turned into zeroes. The order of triangularization is from left to right. Thus, P_j makes the elements below the diagonal in column j equal to zero. The vector w_j that is used in P_j is determined only from column j of the matrix A_{j-1} that is obtained from application of P_1 through P_{j-1} to A. The determination of w_j is very straightforward except for element j and the scaling to make P_j unitary. Elements 1 through j-1 are set to zero for w_j. Disregarding the scaling factor elements j+1 through n are set to be equal to elements j+1 through n of column j of A_{j-1}. The diagonal element is chosen so that the magnitude of element j of column j of $A_j = P_j A_{j-1}$ equals the L_2 norm of column j of A_{j-1}. The sign is taken to be the opposite of the jth diagonal element of A_{j-1}. This choice of element j of the vector w_j makes the sequence of transformations numerically stable, [Wilkinson 65].

Hence,

$$P_j = I - 2w_j w_j^*$$

$$A_{j-1} = P_{j-1} P_{j-2} \cdots P_1 A$$

$$u_k = 0, \quad k = 1, 2, \ldots, j-1$$

$$u_j = (A_{j-1})_{jj} + \text{sign}((A_{j-1})_{jj})d$$

$$uk = (A_{j-1})_{kj} \qquad k = j+1, \ldots, n$$

$$d = \text{sqrt}\left(\sum_{k=j}^{k=n} (A_{j-1})_{kj}^2 \right)$$

$$w_j = u/\text{sqrt}(2d^2 + 2|(A_{j-1})_{jj}|d)$$

If P_j is not explicitly computed but instead $w_j^* A$ computed as one step in forming $P_j A_{j-1}$, then this step computes a linear combination of rows j through n. The only nontrivial computations in computing the linear combination regards columns j+1 through n. Once the linear combination is formed the computations are very similar to Gaussian elimination. Hence, Householders method can, from a computational point of view, be considered as Gaussian elimination extended with a step that instead of chosing a pivot row based on the elements on and below the diagonal in the column (j) to be eliminated as in partial pivoting, computes a linear combination of those rows with a row index between j and n that have non-zero entries in column j.

CONCURRENCY IN HOUSEHOLDER TRANSFORMATIONS

The computation of $v = w_j^* A_j$ can be considered to be the first phase of applying P_j to A_{j-1} with the second phase being the computation of $A_{j-1} - w_j v$, the "elimination" phase.

The first phase includes the computation of a linear combination of rows. Obviously, the different elements of the linear combination can be computed concurrently. Such a scheme requires that the elements of w_j be broadcasted to all the columns. This broadcasting can be avoided by pipelining the computations from column to column as discussed later.

The formation of an inner product between two vectors can be performed in a logarithmic number of steps. Such an implementation is partially instantiated in space (a tree of adders) and partially in time. For simplicity it is here assumed that the linear combination is formed in linear time with one unit.

If it weren't for the scaling factor and the jth element of w_j the linear combination could be formed very nicely with only local communication in an orthogonal two-dimensional array, since the coefficients in the linear combination would be elements j through n in the jth column. If the second term in u_j is omitted at first then the linear combination except for the scaling factor can be formed directly from A_{j-1}. The first element of the linear combination then equals d^2. The scaling factor can now be directly computed and it remains to account for the fact that the linear combination is not yet complete. A factor of the jth row is missing. In the algorithm that follows the missing factor of row j is included in the computations that form the elimination phase.

The elimination phase can be carried out in one step. The linear combination computed in phase 1 has to be broadcasted to accomplish this. Again, pipelining can be used to avoid the broadcasting. Phase 2 of the application of P_j can be pipelined with the first phase of the computations associated with the application of P_{j+1}. Since phase 1 here is constrained to be linear in time (can be made logarithmic), there is no point in instantiating phase 2 fully in space. Hence, the treatment below focus on an implementation assuming linear time also for phase 2.

If the matrix A is a band matrix with r non-zero codiagonals below the main diagonal and s non-zero codiagonals above the main diagonal, then only r+1 rows and s+r+1 columns have to be considered at a time. The matrix R will have r+s non-zero codiagonals, since it is obtained by subtracting linear

combinations of r+1 successive rows from the first row included in the linear combination. The active part of the computations can be illustrated by a window of size (r+1) x (r+s+1) that slides down the diagonal with two of the corners on the diagonal as in Figure 1. Successive windows are displaced by one column.

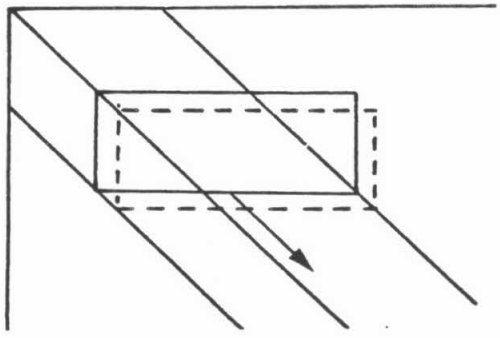

Figure 1: Windows of concurrently active computations

A Concurrent Algorithm for Householder transformations

In the elimination of the nonzero elements below the diagonal in the jth column the rows of A_{j-1} are changed as follows:

$$(A_j)_{j,m} = -\text{sign}((A_{j-1})_{j,j})\left(\sum_{k=j}^{k=n} (A_{j-1})_{k,j}(A_{j-1})_{k,m}\right)/d \qquad (1)$$

$$(A_j)_{i,m}=(A_{j-1})_{i,m}-(A_{j-1})_{i,j}\,\text{sign}((A_{j-1})_{j,j})((A_{j-1})_{j,m}-(A_j)_{j,m})/(d+|(A_{j-1})_{j,j}|)$$

$$i > j, \quad m > j \qquad (2)$$

In these expressions m can be associated with a module in a linear array. This fact may become more clear if the column index instead is written as m+j where $0 \leq m \leq r+s$. Similarly the cyclic time behavior may become more apparent for a band matrix if the summation index k is written as j+k with k being in the range $0 \leq k \leq r$. In the second equation i can also be rewritten as j+i with $0 < i \leq r$ for a band matrix. This notation is convenient when

computations associated with different columns are instantiated in space, whereas computations associated with one column are instantiated in time. The algorithm presented next is based on these assumptions and the space-time trade off is shown in Figure 2.

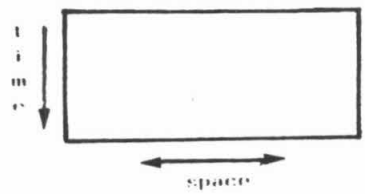

Figure 2: Time-space trade off in equations 1 and 2

The sum of row products in equation (1) has to precede the computations of equation (2). The computation of equation (1) for the next column, j+1, can start one time step after $(A_j)_{j+1,j+m}$ of equation (2) has been computed. It should be noticed that the computations of $(A_j)_{j+1,j+m}$ (equation (1)) can not be completed until all rows $(A_{j-1})_{j+i,j+m}$ (equation (2) for the preceding column) have been computed.

A somewhat more detailed description of a concurrent algorithm for Householder transformations is the following:

1. Compute $p = \sum_{i=0}^{i=r} (A_{j-1})_{j+i,j}(A_{j-1})_{j+i,j+m}$

2. Compute d, $b = -\text{sign}((A_{j-1})_{j,j})$ and $c = |(A_{j-1})_{j,j}|$

3. Compute $e = b/d$, $f = b/(d+c)$

4. Compute $(A_j)_{j+1,j+1+m} = e*p$,

   and $prow = f*((A_{j-1})_{j,j+m} - (A_j)_{j,j+m})$

5. Compute $(A_j)_{j+i,j+m} = (A_{j-1})_{j+i,j+m} + (A_{j-1})_{j+i,j}prow$

In the above algorithm time is associated with indices i and j, and space with index m.

A schematic illustration of how the computations defined by steps 1 through 5 of the algorithm above can be arranged is shown in Figure 3. The leftmost module treats the column that is to be eliminated next (j+1) and the rightmost module column j+r+s. The array is arranged so that the nontrivial part of row j of R, $(A_j)_{j,j+m}$, leaves the array at the bottom.

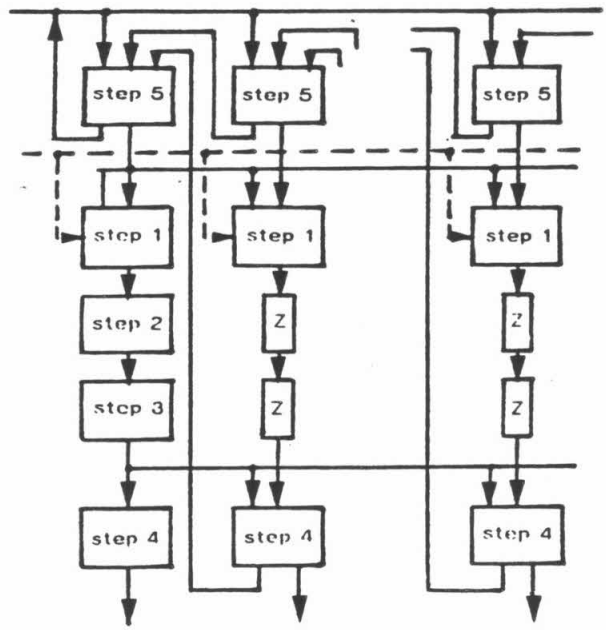

Figure 3: Schematic array for Householder transformation

A COMPUTATIONAL ARRAY

It should be clear from the previous discussion that even though computations associated with two successive windows can be pipelined, the computations of step 1 for column j+1 cannot be completed until step 5 of column j has been completed. To limit the number of multipliers and adders connected in series the computations in step 5 are assumed to lead the computations in step 1 by one time step. For simplicity in the communication needs the new row that has to be included when proceeding from column j to column j+1 is supposed to be treated last in step 1. Hence, step 1 will not be completed until two time

steps after step 5. An additional few steps are required for the computation of the square root, step 2, two intermediate variables, step 3, and the "pivot" row, step 4, before the computations can proceed to step 5.

The sequencing requirements of the algorithm makes it necessary to introduce explicit control. Explicit control is also used to implement the finite accumulation and reset implied by step 1.

A detailed illustration of an array and algorithm implementing steps 1 through 5 is shown in Figure 4. The array computes the matrix R for a band matrix with r+s codiagonals. The application of the matrices P_j to the right hand side (y) of the equation can be performed by a module similar to the "inner" modules of the array in Figure 4. The back substitution part is the same as for Gaussian elimination, see e.g., [Kung 80b] or [Johnsson 81].

The uppermost part of the array performs step 5. The elements $(A_{j-1})_{j+i,j}$ $i = 1,2,..,r$ are broadcasted horizontally. Shift registers are used to store the columns as they are computed. The columns move from right to left. As the computational windows move down the diagonal, old rows leave the array and new ones enter. In Figure 4 the new rows enter below the elimination part. The mechanism for shifting in a new row is not shown in its complete form. It takes r+s+1 time steps to shift a new row into the array if no parallelism in the input is assumed and the time step for shifting in and out data is assumed to be equal to the time step for the rest of the array. In Figure 4 the row is assumed to be in the proper position when needed. The elements of the new column are all zero with the exception of the element that also is in the new row. Thus, the rightmost module is furnished with zeroes for column j+r+s+1.

In order to separate the computations of step 5 and 1 intermediate storage is introduced before the data enters the third row of the array. The third row computes the linear combination of step 1. The idea behind the intermediate storage is to reduce the length of the cycle time of the array. Hence, step 1 trails step 5 by one time step. It takes r+1 cycles to complete the computation of the linear combination.

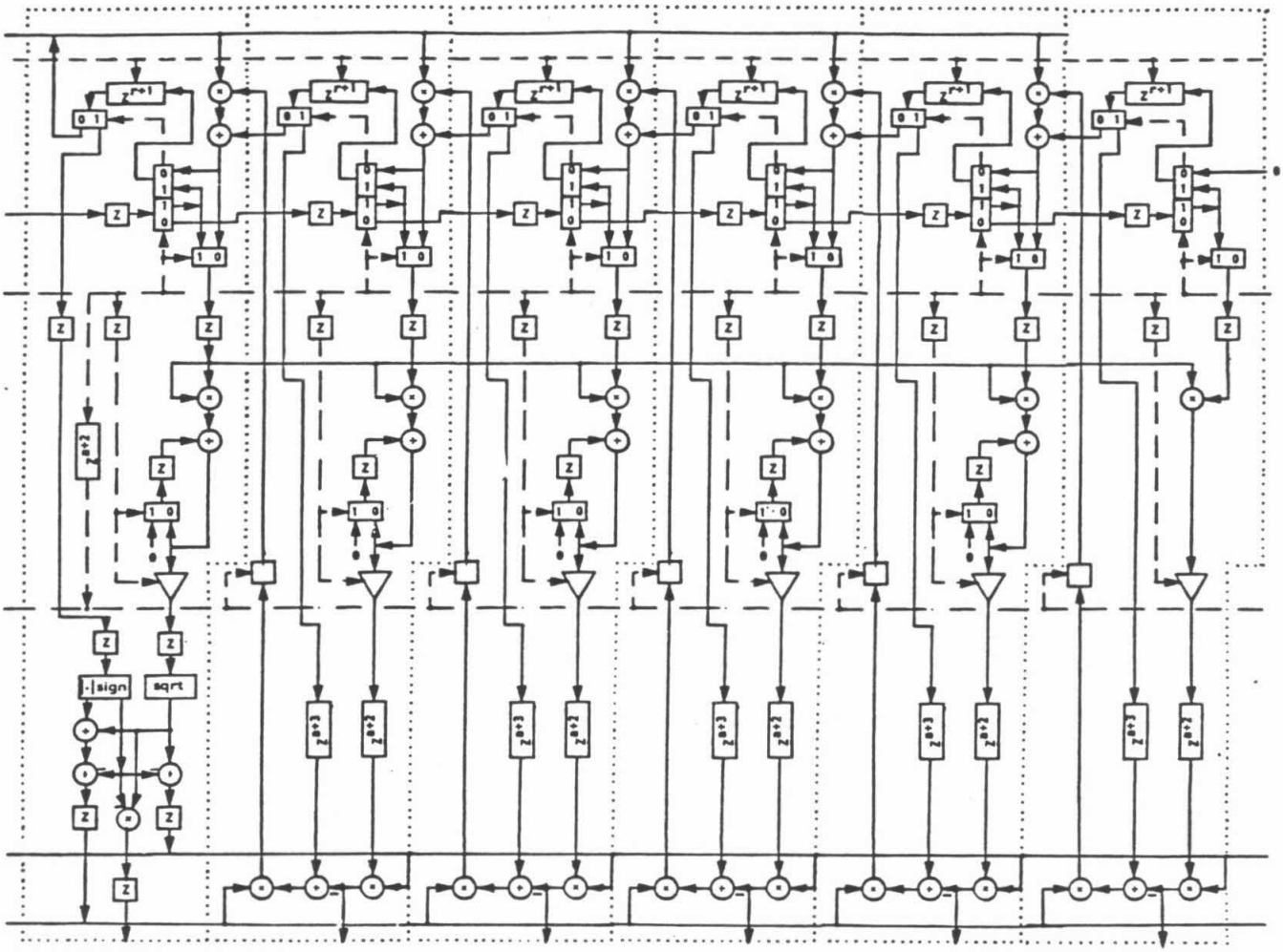

Figure 4: An array performing Householder transformations

The contents of the accumulator in the first column of the array is indeed d^2. This result is latched into a register for intermediate storage to separate the computation of the square root from the final cycle of the accumulation. The result of the square root calculation is passed to logic that performs the computations of step 3. The output of this logic goes to intermediate registers. Unfortunately, most of the array will be idle during the computations of steps 2 and 3. Appropriate delays have to be introduced in parallel signal paths for data to arrive at the the logic for the computation of step 4 at the right time. A corresponding delay has to be introduced in control of the exchange of the "pivot" row. In Figure 4 this delay has been

instantiated fully in space. The same function can be accomplished with sample and hold circuits at the expense of additional control.

The bottom row of the array computes step 4. The "old" row, $(A_{j-1})_{j,j+m}$ is in the first position of the shift registers holding the columns of the computational window, one time step before the linear combination of step 1 is formed, due to the intermediate delay introduced between steps 5 and 1.

Explicit control of the array is necessary. The shift registers holding the columns are shifting during r+1 consecutive time steps and will store the information until the computations associated with the next column can start. With the scheme described above the new row enters the shift registers and the linear combination stage during the last time step of column elimination. (Obviously, no elimination need to be performed on the new row.) The accumulators also needs to be reset once for every column.

The control signal shall once for every column latch the new row into the shift registers instead of the result of the elimination step, and reset the accumulators. This control signal can also be used to control tri-state drivers for the output from the accumulators. The period of the control signals are not only determined by r, the number of codiagonals below the main diagonal, but also by the time, a, required to compute the square root. At the time step 4 is completed the "pivot" row is stored in a row of storage cells. The read/write operation of these storage cells can be controlled by a delayed version of the control signal for the accumulators. Assuming that the input/output operations do not limit the performance a timing diagram as in Figure 5 can be drawn.

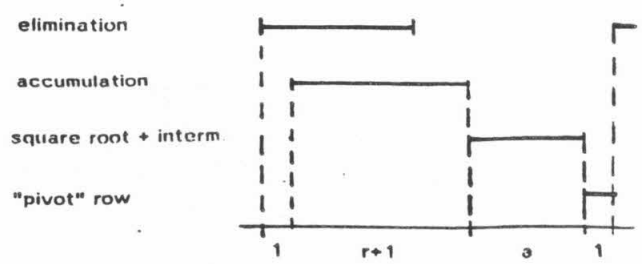

Figure 5: Timing diagram for the array in Figure 4

Several improvements of the array in Figure 4 can be made. For example, of the total period of r+a+3 time steps two are due to intermediate storage. These two steps can be removed at the expense of a longer cycle (for all steps). The third of the three "extra" steps can be removed by introducing extra storage for the "next" row, so that the computation of the linear combination can start with the new row instead of ending with it. As mentioned before the r cycles can be reduced to $\log_2 r$ at the expense of additional hardware.

Pipelining of column operations

In the array of Figure 4 several signals are broadcasted. This broadcasting may well be the limiting factor in determining the minimum cycle time under which the array will operate correctly. The broadcasting can be avoided by introducing intermediate storage between successive columns. The computations associated with different columns become pipelined.

It is necessary to introduce a delay also of the control signals in going from one column to the next. The direction of the control signals is now of importance and should be from left to right. It is also necessary to increase the length of the shift period to r+2 instead of r+1. The reset operation is still made on the last step of the shift sequence within a period. However, it is not necessary to space data out in time as in the case of Gaussian elimination, where for a fully pipelined array three time steps are required to complete one elimination operation, [Johnsson 81].

Partitioning of problems

If the problem is of a size larger than the array can hold partitioning in some way is required. The length of the shift registers in the array in Figure 4 is a function of r, the number of non-zero codiagonals below the main diagonal, whereas the number of modules depends on r+s, the total number of non-zero codiagonals. If the shift registers cannot be made large enough within the array, outside storage is required. The scaling factor cannot be computed until the part of a column that falls within the computational window has been treated. If outside storage is used the adjustment to column length is

entirely made through changing the length of the control signal for accepting column data and resetting the accumulators.

If more modules are required than exist in the array, outside storage is obviously necessary not only for the columns that do not fit within the array, but also for intermediate results. The elements of column j to be eliminated and that are passed from the leftmost module to the right needs to be stored and supplied to the array for each set of columns within the computational window treated by the array. Similarly, the scaling factors need to be stored as well as the elements of the "pivot" row that corresponds to the set of columns being treated. Conversely, it is also necessary to provide means by which these data can be loaded from outside. Essentially only the "inner" modules can be used for sets of columns other than the leftmost set. In order to use the leftmost module it would be necessary to include logic to bypass the computations of steps 2 and 3, and to introduce logic that can perform step 4.

CONCLUSIONS

An algorithm and array for the QR-method based on Householder transformations has been proposed. The array in its simplest form computes the matrix R and the corresponding transformations on the right hand side of the equation in $(r+3+a)n$ time steps but suffers from broadcasting. The parameter a corresponds to the number of multiply/add times required to compute the square root function and the parameter n to the size of the matrix. By fairly simple means broadcasting can be avoided without changing the number of time steps. The number of steps can be reduced by 2n at the expense of increased cycle time. At the expense of substantially more hardware r can be replaced by $\log_2 r$ in the above expression.

Whether the reduction from r to $\log_2 r$ time steps for performing the required computations on a column is of interest or not, as well as the performance penalty incurred by the square root computation, is largely dependent on how data input and output is handled. With sequential input and output r+s+1 time steps are required for loading/unloading of rows. Input output operations can be performed concurrently with the computations of the algorithm.

It is interesting to compare the number of steps required to compute the QR-factorization through Householder transformations with the number of steps required for LU-decomposition through Gaussian elimination. The latter method requires (r+3)n to (2r+3)n steps when partial pivoting is used in a pipelined array. Depending on the relative values of r and a, it may well be that factorization through Householder transformations can be performed faster. Also, the control structure for the latter method is much simpler and the numerical properties clearly superior.

ACKNOWLEDGMENT

The author would like to thank Professors Heinz Otto Kreiss and Bengt Fornberg of the Applied Mathematics Department of Caltech, who pointed out the importance of the QR-method based on Householder transformations in large scale scientific computing and thereby initiated this study.

The author gratefully acknowledges the support provided by the Defense Advanced Research Project Agency. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion of DARPA, the U.S. Government, nor any person or agency connected with them.

REFERENCES

[Dahlquist 74]
Dahlquist G., Bjorck A., Anderson N.
Numerical Methods.
Prentice-Hall, 1974.

[Gentleman 73]
Gentleman, S. Morven.
Least Squares Computation by Givens Transformations without Square Roots.
J. Inst. Maths. Applics. 12:329-336, 1973.

[Hammarling 74]
Hammarling, S.
A Note on Modifications to the Givens Plane Rotation.
J. Inst. Maths. Applics. 13:215-218, 1974.

[Heller, Ipsen 81]
Heller, Don E., Ipsen, Ilse I. P.
Systolic Networks for Orthogonal Decompositions with Applications.
Technical Report CS-81-18, Computer Science Dept., Pennsylvania State University, University Park, August, 1981.

[Isaacson 66]
Isaacson, E., Keller, H.
Analysis of Numerical Methods.
Wiley, 1966.

[Johnsson 81]
Johnsson, L. Computational Arrays for Band Matrix Equations.
Technical Report 4287, Caltech Computer Science Department, May, 1981.

[Kung, Leiserson 80]
Kung, H.T. and Leiserson, Charles E.
Algorithms for VLSI Processor Arrays. In Introduction to VLSI Systems. Addison-Wesley, 1980.
Mead, Carver A. and Conway, Lynn A.

[Kung 80]
Kung, S.Y.
VLSI Matrix Computation Array Processor.
In MIT Conference on Advanced Research in Integrated Circuits.
MIT, February, 1980.

[Wilkinson 65]
Wilkinson, J.
The Algebraic Eigenvalue Problem.
Oxford, 1965.